MY472 - Week 4
XML, RSS, and Advanced Scraping

Friedrich Geiecke
18 October 2021
Introduction

- Last week we discussed some examples of scraping tables or simple unstructured content

- To scrape some websites e.g. with forms or dynamic elements, we need more advanced tools

- This week we will discuss XML, RSS, and XPath, and use RSelenium for browser automation
Plan for today

- XML
- RSS
- XPath
- Scraping with (R)Selenium
- Guided coding session
XML

- XML = eXtensible Markup Language
- XML: Store and distribute data
- HTML: Display data
- XML looks a lot like HTML, but is more flexible (no predefined tags, author can invent tags to structure document)

Reference and further information: https://www.w3schools.com/xml/xml_whatis.asp
<?xml version="1.0" encoding="UTF-8"?>
<courses>
    <course>
        <title>Data for Data Scientists</title>
        <code>MY472</code>
        <year>2020</year>
        <term>Michaelmas</term>
        <description>A course about collecting, processing, and storing data.</description>
    </course>
    <course>
        <title>Computer Programming</title>
        <code>MY470</code>
        <year>2020</year>
        <term>Michaelmas</term>
        <description>An introduction to programming.</description>
    </course>
</courses>
XML, Example 2 (with DTD)

```xml
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>The next assignment will be due on 29 October at 2pm.</body>
</note>
```

- This XML has a DTD (Document Type Definition)
- DTD is a schema language with relatively limited capabilities, XML Schema has more features
Steps in XML parsing in R

1. Parse an XML file with `read_xml()` in `xml2` package
2. Select nodes with `xml_nodes()`
3. Extract text using `xml_text()`

Analogous for HTML with `read_html()`, `html_nodes()`, and `html_text()` (last week)
Further XML examples

- Canadian members of parliament: https://www.ourcommons.ca/Members/en/search -> select “Export as XML”


- epub (books)

- Office documents (OpenOffice, MS)

- RSS (web feeds -> next topic): http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907
RSS
RSS

- Really Simple Syndication
- Written in XML
- RSS feeds allow users to see new contents from a range of websites quickly and in one place
- RSS aggregators gather and sort RSS feeds
- RSS feed example: The Guardian RSS feed (more in the guided coding session)
Imaginary RSS feed

Based on: https://www.w3schools.com/xml/xml_rss.asp
XPath
Selecting XML/HTML nodes with XPath

- Last week we discussed CSS selectors to select elements, XPath offers another way
- Both XML and HTML document have a tree structure
- **XPath** (or XML Path Language) is a syntax for defining parts of the tree/document
- Can be used to navigate through elements and attributes
Types of XPath

- Relative XPath: //div[2]/p[1]
Our favourite website

<!DOCTYPE html>
<html>
  <head>
    <!-- CSS start -->
    <style>
      .text-about-web-scraping {
        color: orange;
      }
      .division-two h1 {
        color: green;
      }
    </style>
    <!-- CSS end -->
  
  </head>
  <body>
    <div>
      <h1>Heading of the first division</h1>
      <p>A first paragraph.</p>
      <p>A second paragraph with some <b>formatted</b> text.</p>
      <p class="text-about-web-scraping">A third paragraph now containing some text about web scraping ...</p>
    </div>
    <div class="division-two">
      <h1>Heading of the second division</h1>
      <p>Another paragraph with some text.</p>
      <p class="text-about-web-scraping">A last paragraph discussing some web scraping ...</p>
    </div>
  </body>
</html>
In more detail: Some basic syntax (1/2)

- `/`: Selects from the root node, e.g. `/html/body/div[2]/p[1]`

- `//`: Selects specific nodes from the document, e.g. `//div[2]/p[1]`

- `//div/*`: Selects all nodes which are immediate children of a div node

- `//div/p[last()]`: Selects the last paragraph nodes which are children of all div nodes
In more detail: Some basic syntax (2/2)

- //div[@*]: Selects all division nodes which have any attribute

- //div[@class]: Selects all division nodes which have a class attribute

- //div[@class='division-two']: Selects all division nodes which have a class attribute with name “division-two”

- //*[@class='division-two']: Selects any node with a class attribute with name “division-two”

- etc.

Reference and full details: [https://www.w3schools.com/xml/xpath_syntax.asp](https://www.w3schools.com/xml/xpath_syntax.asp)
**Comparison: XPath vs CSS selector**

<table>
<thead>
<tr>
<th>Selector type</th>
<th>CSS selector</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>By tag</td>
<td>&quot;h1&quot;, &quot;p&quot;</td>
<td>&quot;/h1&quot;, &quot;/p&quot;</td>
</tr>
<tr>
<td>By class</td>
<td>&quot;.division-two&quot;</td>
<td>&quot;//*[@class='division-two']&quot;</td>
</tr>
<tr>
<td>By id</td>
<td>&quot;#exemplary-id&quot;</td>
<td>&quot;//*[@id='exemplary-id']&quot;</td>
</tr>
<tr>
<td>By tag with class or id</td>
<td>&quot;div.division-two&quot;</td>
<td>&quot;//div[@class='division-two']&quot;</td>
</tr>
<tr>
<td>Tag structure (p as a child of div)</td>
<td>&quot;div &gt; p&quot;</td>
<td>&quot;//div/p&quot;</td>
</tr>
<tr>
<td>Tag structure (p which is a second child of the div node with class name division-two)</td>
<td>&quot;div.division-two &gt; p:nth-of-type(2)&quot;</td>
<td>&quot;//div[@class='division-two']/p[2]&quot;</td>
</tr>
</tbody>
</table>

- Guide: [https://ghostinspector.com/docs/css-xpath-conversion/](https://ghostinspector.com/docs/css-xpath-conversion/)
Scraping with RSelenium
Why?

- Scenario 3
- Many websites cannot be scraped as easily as in scenarios 1 & 2 for various reasons
  - Form
  - Authentication
  - Dynamic contents
Selenium

- https://www.selenium.dev/
- A technology for browser automation
- General idea: **Browser control** to scrape dynamically rendered web pages
- Originally developed for web testing purposes
- **RSelenium**: An R binding for Selenium
  - Launch a browser session and all communication will be routed through that browser session
Selenium drivers

There are two general strategies to run this

1. Normal browsers
   - Chrome
   - Firefox
   - etc.

2. Headless browser (will not display website)
   - Allows to set up the browser in a situation where you do not have a visual device (i.e. Crawler on the cloud) or do not need an open browser window
   - Common headless browser: phantomJS
   - Selenium in Python e.g. allows to also run Chrome or Firefox in headless mode
Some key functions (1/2)

- **RSelenium package**

  ```r
  library("RSelenium")
  ```

- **Create browser instance with**

  ```r
  rD <- rsDriver(browser = c("firefox"))
  driver <- rD$client
  ```

- **Navigate to url**

  ```r
  driver$navigate("https://www.lse.ac.uk/")
  ```

- **Find element**

  ```r
  some_element <- driver$findElement(using = "xpath", value = "...")
  ```
Some key functions (2/2)

- Click on element

```r
some_element$clickElement()
```

- Type text into box/element

```r
search_box <- driver$findElement(using = "xpath", value = "...")
search_box$sendKeysToElement(list("some text"))
```

- Press enter key

```r
search_box$sendKeysToElement(list(key = "enter"))
```
An exemplary Google search

Let us look at a simple example of \texttt{RSelenium}

```r
library("RSelenium")

rD<- rsDriver(browser=c("firefox"))
driver <- rD$client

url <- "https://www.google.com/"
driver$navigate(url)

xpath_of_search_field <- "..."

search_box <- driver$getElement(using = "xpath", value = xpath_of_search_field)
search_box$sendKeysToElement(list("my472 lse"))

Sys.sleep(1)

search_field$sendKeysToElement(list(key = "enter"))
```
Guided coding session
Markdown files this week

- 01-newspaper-rss.Rmd
- 02-introduction-to-selenium.Rmd
- 03-selenium-lse.Rmd